

## سید محمد طاها طباطبایی – تمرین سری ششم

۹۸۱۲۷۶۲۸۳۸

### چکیده:

در پارت اول، به بررسی فضای رنگی HSI و تبدیل و مقایسه آن با RGB می‌پردازیم. در پارت بعدی، ۳ نمونه فضای رنگی جدید معرفی شده است.

در بخش دوم، اثر سطح‌بندی تصویر و استفاده از بیت‌های کمتر از ۸ در نمایش تصاویر خاکستری و رنگی بررسی شده است. در انتها نیز به بررسی تاثیر نمایش تصویر با تعداد حالت‌های رنگی متفاوت و وزن‌دهی متفاوت به سطح‌ها در کانال‌های رنگی متفاوت پرداخته شده است.

## توضیحات فنی:

- توضیح جزییات پیاده‌سازی توابع در محل پیاده‌سازی کد، به صورت کامنت و داکيومنت نوشته شده است.

### ۶.۱.۱

در سلول اول، با استفاده از تابع RGB2HSI، تصویر را از فضای رنگی RGB به HIS منتقل می‌کند. برای پیدا کردن هر کدام از سه مشخصه، hue، saturation و intensity از توابعی استفاده شده است که طبق فرمول معرفی شده در اسلاید های درس، محاسبات را انجام می‌دهند. در گام اول، اگر از سه کانال رنگی RGB برای بررسی استفاده کنیم، متوجه می‌شویم میزان حضور رنگ قرمز در تصویر بیشتر است. حضور رنگ قرمز بیشتر از ۲ رنگ دیگر، به این معنی است که احتمالا در نمایش فام، باید پیکسل های بسیار سیاه یا سفید بیشتری داشته باشیم، چون در نمایش فام، زاویه های نزدیک به صفر یا ۳۵۹ درجه به رنگ قرمز اختصاص یافته است، و بر اساس اینکه در آن محل، رنگ قرمز بیشتر با سبز ترکیب شده یا آبی، زوایا به صفر یا ۳۵۹ نزدیکتر خواهند بود.

$$\theta = \cos^{-1} \left\{ \frac{12[(R-G)+(R-B)]}{[(R-G)^2 + (R-B)(G-B)]^{1/2}} \right\} \quad (7-17)$$

The saturation component is given by

$$S = 1 - 3(R+G+B)[\min(R,G,B)] \quad (7-18)$$

Finally, the intensity component is obtained from the equation

$$I = 13(R+G+B) \quad (7-19)$$

مشخصه I در فضای HIS، نشان دهنده بیشترین اطلاعات از تصویر است. در تصویر باند I، نقاط روشن تر، نقاطی هستند که شدت رنگ بیشتری داریم، و نقاط تیره تر، نقاطی است که مجموع مقادیر پیکسل های رنگی در آن نقاط کم است.

مشخصه S، اشباع را نشان می‌دهد. اشباع، میزان خلوص رنگ در فضای رنگی است. به طوری که هرچه مقدار حضور یک رنگ، بیشتر از مقدار حضور رنگ های دیگر باشد، اشباع بیشتری داریم. به طور مثال لبه کلاه، چون یک ناحیه است که از ترکیب تقریباً یکسانی از سه رنگ تشکیل شده است، اشباع کمی دارد، برای همین به رنگ تیره تر تبدیل شده. (figure 1)

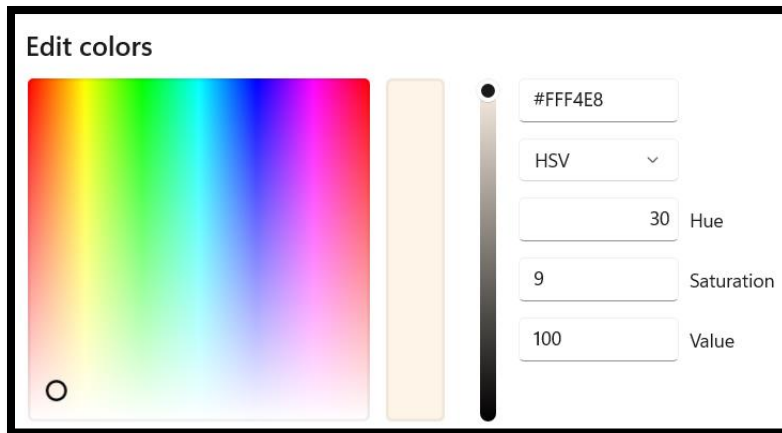


Figure 2 – فام و اشباع کم، درحالی که شدت روشنایی زیاد است.

برخی نقاط تیره، در شاخصه اشباع مقدار بسیار سفیدی پیدا کرده اند، زیرا علی رغم اینکه مقدار intensity زیادی ندارند و به ظاهر تیره هستند، اما چون اشباع رنگ آبی آن ها زیاد است، در شاخصه اشباع پیکسل های روشنی را به خود اختصاص داده اند. (figure 2)

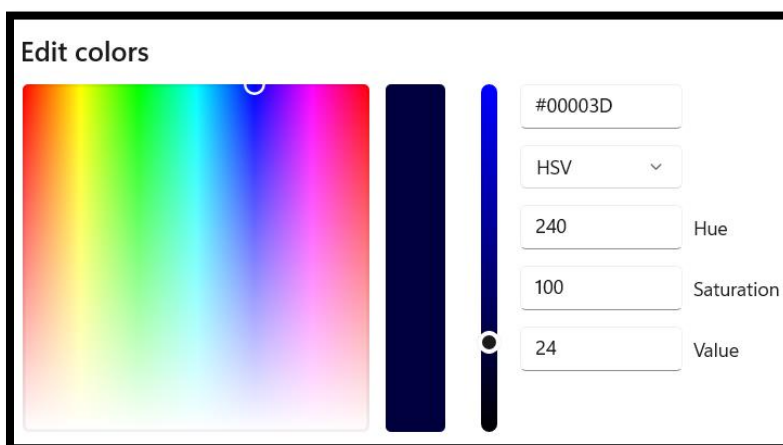


Figure 2 – اشباع ۱۰۰ درصدی با رنگ آبی تیره

مشخصه h، فام را مشخص می کند. نقاط تیره در نتیجه نمایش داده شده، نقاط قرمزی هستند که زوایای نزدیک به صفر دارند، و نقاط روشن، نقاط قرمز نزدیک به زاویه ۳۵۹. مقادیر دیگر نیز به همین ترتیب، از رنج ۰ تا ۳۵۹ به ۰ تا ۲۵۵ مپ می شوند. به طور مثال در قسمت پر روی کلاه که رنگ آبی دارد، رنگ خاکستری روشن داریم که تقریباً همان محدوده رنگ آبی است. قسمت پوست رنگ های تیره است، که به دلیل وجود رنگ قرمز و زرد است. اینکه در این تصویر رنگ های خاکستری میانی زیادی نداریم و شاید تصویر کمی باینری سفید و سیاه است، به این دلیل است که رنگ های میانی، نماینده رنگ سبز و فیروزه ای است که در این تصویر، این رنگ ها کمتر مشاهده می شود.

## ۶.۱.۲

### HWB

مخفف رنگ (hue)، سفیدی (whiteness) و سیاهی (blackness) است. مانند HSL، رنگ می تواند هر جایی در محدوده ۰ تا ۳۶۰ باشد. دو آرگومان دیگر، میزان رنگ سفید یا سیاه را تا ۱۰۰٪ کنترل می کنند (که منجر به یک رنگ کاملاً سفید یا کاملاً سیاه می شود). اگر مقادیر مساوی سفید و سیاه باهم میکس شوند، رنگ حاصل، خاکستری است. ما می توانیم این را شبیه به ترکیب کردن رنگ در نظر بگیریم، که برای ایجاد پالت های رنگی تک رنگ، مفید است.

### Adobe RGB

رنگ ها و یا تون های رنگی مختلف بیشتری نسبت به RGB دارد. در مقایسه با RGB این فضای رنگی ۳۵٪ محدوده رنگی بیشتری دارد البته برخی منابع ادعا می کنند محدوده رنگ های Adobe RGB شبیه به RGB (بیش از ۱۶ میلیون رنگ) است و تنها تفاوت این دو در تعداد رنگ هایی است که هر کدام از آن ها می توانند نمایش دهند. Adobe RGB فضای رنگی ایده آل برای عکاسی است.

### LCH

مخفف lightness، chroma و hue است. مانند LAB، درصد روشنایی می تواند تا ۱۰۰ باشد. مشابه HSL، رنگ یا hue می تواند محدوده ای بین ۰ تا ۳۶۰ باشد. کروما میزان رنگ را نشان می دهد، و ما می توانیم آن را شبیه به اشباع در HSL در نظر بگیریم. اما chroma می تواند از ۱۰۰ تجاوز کند و در واقع، از نظر تئوری نامحدود است. استفاده از LCH به ما امکان دسترسی به طیف وسیع تری از رنگ ها را می دهد.

## ۶.۲.۱

برای پیاده‌سازی قطعه‌بندی از تابع  $quantize$  استفاده شده است. ورودی این تابع، تصویر و تعداد بیت‌های در دسترس برای سطح‌بندی است. این تابع، با تقسیم ۲۵۶ بر  $2^{n\_bits}$ ، تعداد دسته‌های مورد نظر را محاسبه می‌کند. سپس پیکسل‌های تصویر را بر این مقدار تقسیم و به پایین گرد می‌کند. در نهایت حاصل را دوباره در  $coeff$  ضرب می‌کنیم. منطق این عملیات این است که ابتدا پیکسل‌ها به اعدادی بین ۰ تا  $level-1$  مقدار می‌دهد که می‌خواهیم نرمالایز می‌شوند، و در ادامه دوباره در  $coeff$  ضرب می‌کنیم تا به اعداد ابتدای بازه‌ها مپ شوند. برای ۶۴ سطح، به ۶ بیت، ۳۲ سطح به ۵ بیت، ۱۶ سطح به ۴ بیت و ۸ سطح به ۳ بیت نیاز داریم. ۴ تصویر حاصل را نمایش دادیم. مشخص است که تفاوت چشم‌گیری بین تصویر اصلی و تصاویر با ۶۴، ۳۲ یا ۱۶ سطح خاکستری وجود ندارد و چشم ما این تفاوت را تشخیص نمی‌دهد. فقط در تصویر آخر تغییر کیفیت تصویر قابل لمس است.

$L = 64$	$L = 32$	$L = 16$	$L = 8$	
42.663301	35.803170	29.163439	23.169236	PSNR
3.521683	17.090836	78.838150	313.441666	MSE

نتایج مقایسه نشان می‌دهد که MSE در تصویر ۸ سطحی نسبت به بقیه تصاویر یک جهش در مقدار تفاضل دارد که با نتایج شهودی به دست آمده نیز منطبق است. کمترین مقدار PSNR نیز برای همین تصویر به دست آمد.

## ۶.۲.۲

برای سطح‌بندی تصاویر رنگی، باید تابع سطح‌بندی را روی هر کانال تصویر به طور مجزا اعمال کنی. کانال قرمز و سبز را در ۳ بیت و کانال آبی را در ۲ نشان دادیم. به طور مشخص، کیفیت سایه‌های کاهش محسوسی داشته. همچنین در بعضی قسمت‌ها رگه‌های رنگ آبی پدید آمده که به دلیلی این است که رنگ آبی در یک بیت کمتر نمایش داده می‌شود و گاهی نسبت به پیکسل‌های قرمز و سبز اطراف خود به سطح بالاتری گرد شده است، مانند رگه آبی روی بازو مدل.

## ۶.۲.۳

با توجه به اینکه می‌توان برای حالت‌های ۳۲ رنگی و ۱۶ رنگی، حالت‌های مختلفی متصور شد که در هر کدام، تعداد بیت متفاوتی به کانال‌های رنگی اختصاص داد، تمام این حالات را محاسبه کردیم. در حالت ۳۲ رنگی، ۳ تصویر خروجی، هر کدام نمایش‌دهنده حالتی هستند که یک کانال رنگی با ۱ بیت و دو کانال رنگی با ۲ بیت سطح‌بندی شده است. در هر کدام از تصاویر، کانالی که به آن یک بیت اختصاص یافته است، آن رنگ را ضعیف‌تر نمایش می‌دهد. به طور مثال در تصویری که کانال آبی ۱ بیتی است، تصویر به سمت رنگ زرد متمایل است. تصویری که ۱ بیت قرمز دارد، فیروزه‌ای رنگ و تصویری که ۱ بیت سبز دارد، بنفش‌تر به نظر می‌رسد.

مشابه حالت بالا، برای نمایش ۱۶ رنگ متفاوت نیز، ۳ تصویر با ۳ تقسیم بیت متفاوت برای کانال‌های رنگی داریم. در اینجا نیز، تصویری که ۲ بیت برای رنگ قرمز دارد، قرمزتر، تصویری که ۲ بیت برای سبز دارد، سبزتر و تصویر آخر نیز به دلیل اختصاص ۲ بیت برای رنگ آبی، آبی‌تر دیده می‌شود.

در حالت ۸ رنگه نیز، به هر کانال رنگی ۱ بیت اختصاص داده ایم.

```

import numpy as np
import math
import cv2
import matplotlib.pyplot as plt
from skimage.metrics import peak_signal_noise_ratio

def RGB2HSI(image):
    """
    Transform image from RGB space to HSI. Calculates Hue, Saturation &
    Intensity.

    Inputs:
        - image: The source image.

    Returns:
        Image in HSI space.
    """

    # suspend errors: 1) divide by zero 2) invalid data-type
    with np.errstate(divide='warn', invalid='warn'):

        # normalize pixels in range [0,1]
        image_normal = np.float32(image)/255.0

        # Separate color channels
        # typr:ignore
        red = image_normal[:, :, 0]
        # type: ignore
        green = image_normal[:, :, 1]
        # type: ignore
        blue = image_normal[:, :, 2]

        h = hue(red.copy(), green.copy(), blue.copy())

        s = saturation(red.copy(), green.copy(), blue.copy())

        i = intensity(red.copy(), green.copy(), blue.copy())

        return h, s, i

def hue(red, green, blue):

```

```

"""
    Calculate the hue( $H$ ) in the HSI color space from the RGB channels.
Hue is calculated
    according to the following formula.


$$H = \begin{cases} \theta & \text{if } (B \leq G) \\ 360 - \theta & \text{else} \end{cases}$$


    where:


$$\theta = \arccosine \left\{ \frac{1/2 * [(R - G) + (R - B)]}{\sqrt{\text{power}((R - G), 2) + (R - B) * (G - B)}} \right\}$$


    Inputs:
        - red: Red channel in RGB space.
        - green: Green channel in RGB space.
        - blue: Blue channel in RGB space.

    Returns:
        The hue.

"""

# numerator of equation
numerator =
np.multiply(0.5, (np.add(np.subtract(red, green), np.subtract(red, blue))))

# denominator of equation
z1 = np.add(np.power(np.subtract(red, green), 2),
np.multiply(np.subtract(red, green), np.subtract(green, blue)))
z1 = np.where(((z1-0) <= 0.00001) , 0.00001 , z1)
denominator = np.sqrt(z1)

# final division
x = np.divide(numerator, denominator)

```



```

# tetha in range [0,2*pi]
tetha = np.arccos(np.divide(np.multiply(x,(math.pi*2)), 360))

# if B <= G  h = tetha , else h = 2*pi - tetha
hue = np.where(blue<=green , tetha , (math.pi*2.0 - tetha))

return hue

def saturation(red, green, blue):
    """
    Calculate the saturation in the HSI color space from the RGB channels.
    Saturation is
    calculated according to the formula below.


$$S = 1 - ( 3 * [\min( R, G, B)] / (R + G + B) )$$


    Inputs:
        - red: Red channel in RGB space.
        - green: Green channel in RGB space.
        - blue: Blue channel in RGB space.

    Returns:
        The saturation.

    """
    minimum = np.minimum(np.minimum(red, green), blue)
    saturation =
np.subtract(1,np.multiply(np.divide(3,np.add(np.add(red,blue,green),0.0001
)),minimum))
    # saturation = 1 -
np.multiply(np.divide(3,np.add(red,blue,green)),minimum)

    return saturation

def intensity(red, green, blue):
    """
    Calculate the intensity in the HSI color space from the RGB channels.
    Intensity is
    the average value of the three channels.

```

Inputs:

- red: Red channel in RGB space.
- green: Green channel in RGB space.
- blue: Blue channel in RGB space.

Returns:

The Intensity.

"""

```
return np.divide((blue + green + red), 3)
```

```
def normalize(array,newMax,newMin):
```

"""

A simple normalization function.

Inputs:

- array: Array to be normalized
- newMax: Max of new range.
- newMin: Min of new range.

Returns:

Normalized array.

"""

```
if isinstance(array, list):
```

```
    return list(map(normalize, array,newMax,newMin))
```

```
if isinstance(array, tuple):
```

```
    return tuple(normalize(list(array),newMax,newMin))
```

```
    normalizedData = (array-np.min(array))/(np.max(array)-  
np.min(array))*(newMax-newMin) + newMin
```

```
    return normalizedData
```

```
def quantize(array, n_bits):
```

"""

Image array(range of 0 to 255) is quantized to the desired number of bits. Its actually generate

a new array, assuming that the representation uses 'n\_bits' instead of the default 8 bits.

Inputs:

- array: The image matrix
- n\_bits: Given number of bits.

```

Returns:
    The quantized array.
"""
coeff = 2**8 // 2**n_bits
return (array // coeff) * coeff

def mean_square_error(imageSource, imagetarget):
    """
    The "Mean Squared Error" between the two images is the
    sum of the squared difference between the two images.
    the lower the error, the more "similar" the two images are.

    NOTE: the two images must have the same dimension

    Inputs:
        - imageSource: The source image, we want to calculate the target
        image difference of
        - imageTarget: The target image, we calculate how far it is from
        the source

    Returns:
        The MSE
    """

    # cumulative difference
    err = np.sum((imageSource.astype("float") -
    imagetarget.astype("float")) ** 2)

    # divide by length*width
    err /= float(imageSource.shape[0] * imageSource.shape[1])

    return format(err, '.6f')

def PSNR(srcImage, testImage):
    """
    Implementation of 'Peak Signal to Noise Ratio' method
    using, sci-kit image library.
    The greater the result, the more "similar" the two images are.

    Inputs:

```

```
- srcImage: The source image, we want to calculate the target  
image difference of  
- testImage: The target image, we calculate how similar it is with  
the source
```

```
Returns:
```

```
    The PSNR
```

```
"""
```

```
return peak_signal_noise_ratio(srcImage,testImage)
```